ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-783

Proactive and Reactive Reconfiguration for the Robust Execution of Multi Modality Plans

Enrico Scala[1] and **Pietro Torasso**[1]

Abstract. The paper addresses the problem of executing a plan in a dynamic environment for tasks involving constraints on consumable resources modeled as numeric fluents. In particular, the paper proposes a novel monitoring and adaptation strategy joining reactivity and proactivity in a unified framework. By exploiting the flexibility of a multi modality plan (where each action can be executed in different modalities), reactivity and proactivity are guaranteed by means of a reconfiguration step. The reconfiguration is performed (i) when the plan is no more valid to recovery from the impasse (reactively), or (ii) under the lead of a kernel based strategy to enforce the tolerance to unexpected situations (proactivity). Both mechanisms have been integrated into a continual planning system and experimentally evaluated over three numeric domains, extensions of planning competition domains. Results show that the approach is able to increase the percentage of cases successfully solved while preserving efficiency in most situations.

1 INTRODUCTION

The execution of plans in realistic environments has to face a number of challenges: in such dynamic environments it is hard to make accurate predictions (in particular for what concerns the resource profile), and the occurrence of exogenous events could make the actual state very different from the one predicted at planning time.

To deal with this problem, the continual planning paradigm is receiving increasing attention ([9, 4]): in such an approach an agent has the capability to interleave execution and (re)planning all along the plan execution in order to recognize and handle all those situations where discrepancy invalidates the current plan of actions. The continual planning paradigm, and in particular the replanning mechanism, is necessary in all those domains where anticipating all the possible contingencies at planning time is not feasible (e.g., via conformant or contingent plans) for either computational reasons or impossibility to find solutions because of the incompleteness of the domain knowledge [9].

In replanning, a critical aspect concerns the possibility of finding a repair plan in an efficient way. In fact, the on-line setting imposes strict constraints on the amount of computational resources (especially time). Some recent papers ([5],[15],[8]) have shown that, for *small* deviations from the nominal behavior, the plan repair problem can be efficiently managed without the necessity of replanning from scratch. While this represents a big step ahead, most of the work has been mainly focused on the maintenance of propositional conditions. Limited attention has been played to deal with numeric fluents ([6]), which instead play a crucial role for realistic domains where consumable resources have to be handled.

[1] Dipartimento di Informatica - Universita' degli Studi di Torino (Italy), email: {scala,torasso}@di.unito.it

In this paper we present a new approach for the execution and the adaptation of plans involving numeric fluents. This approach is aimed at limiting the need of replanning as much as possible. In particular we exploit the notion of Multi Modality Action recently introduced in [12], which allows to model the way the different execution modalities of the same action impact on the usage of resources, modeled via numeric fluents. The paper shows that in many cases an impasse in the plan execution can be solved by changing the execution modalities of the actions still to be performed (without the need of a replanning step). This reconfiguration step is started after the occurrence of an impasse in the plan execution in a reactive strategy. The paper presents also a pro-active approach where the agent try to prevent an impasse by anticipating the change in some execution modalities. In particular, the paper proposes a kernel based method ([13]) for selecting the execution modality of the next action to be performed. In such a way the system avoids possible reconfiguration/replanning steps, or at least increases the recovery power. In many cases, in fact, resources are limited and cannot be renewed, so anticipation is necessary to prevent the agent to be trapped in dead-end situations.

Section 2 provides a formalization of the notion of Multi Modality Action and the reconfiguration problem. The reactive strategy which makes use of the reconfiguration for solving plan execution failure is described in Section 3, while the kernel based method and the proactive strategy are presented in Section 4 and 5, respectively. An extensive experimental setup is reported in Section 6.

2 BACKGROUND

This section introduces the planning language of reference and formalizes the problem we are interested in. According to the PDDL 2.1 ([6]) terminology, the domain in our system is modeled via a set F of propositional fluents and a set X of numerical fluents representing the qualitative and quantitative properties of domain objects, respectively. A state s is a pair $< F(s), X(s) >$ where $F(s) \subseteq F$ asserts which propositional fluents are true in s, while $X(s)$ is an assignment of real values to all the numeric fluents in X. As hinted at in the introduction, the Multi Modality Action (MMA) formalism is intended to model the different ways a given action can be performed. In particular an execution modality impact (in numeric terms) the resource profiles. For this reason the MMA splits the knowledge about the preconditions and effects of the action at the propositional level (qualitative behavior) from the preconditions and effects of the numeric one (quantitative behavior), which are implied by the modality selection. More formally:

Definition 1 (Multi-Modality Action) *A Multi-Modality Action (MMA) a is the tuple $< Pre_{prop}(a), Eff_{prop}(a), mods(a) >$ where:*

- *Pre_{prop} is a set of propositions defined over F modeling the applicability conditions (in propositional terms) for a.*
- *Eff_{prop} is a set of propositions defined over F, expressing the effect of the application of a (as typical in PDDL language we may have both positive and negative effects).*
- *$mods$ is a collection of modalities. Each modality m defines a specific way of performing a, and is modeled as a pair $< Pre_{num}, Eff_{num} >$ where:*
  - *Pre_{num} is a set of comparisons specifying the preconditions of the execution of MMA a in modality m (Each comparison has the form $(exp, \{<, <=, =>, >\}, exp')$.*
  - *Eff_{num} is a set of numeric operations. Each numeric effect is the triple (f,op,exp), where f is the numeric fluent affected by the operation, op is one of $\{+ =, - =, =\}$. Eff_{num} represents the way the action a changes the world state if executed in modality m.*

The terms exp and exp' denote numeric expressions defined over real constants and over the set X of numeric fluents[2].

An MMA a is executable in modality m in a state s when both the propositional part of the state s (denoted as s_{prop}) satisfies $Pre_{prop}(a)$ and the numeric part of the state s (denoted as s_{num}) satisfies $Pre_{num}(a(m))$. Let a be executable in s with modality m, its application to s produces a successor state s' where: $s'_{prop} = (s_{prop} \cup Eff^{+}_{prop}(a)) \backslash Eff^{-}_{prop}(a)$ and each numeric fluent f occurring in s_{num} of $a(m)$ is modified according to *op* and exp.

Note that, given a state s, an action a could be executed in more than one modality. This is the main characteristic that gives us the flexibility to decide the modality, given the actual context.

A Multi Modality Plan (MMP) $\pi = a_0, a_1, .., a_{n-1}$ is a sequence of MMAs. We will denote with $\pi^c = a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})$ a configured plan where each a_i is associated with a specific modality m_i such that $m_i \in mod(a_i)$

Given an initial state s_0, a set G of goal conditions[3] and a set of MMAs, we can say that $\pi^c = a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})$ is valid for s_0 and G iff the state s_n, predicted by using π^c, satisfies G. Note that each action $a_i(m_i)$ in π^c must be applicable in the state s_i generated by its predecessor $a_{i-1}(m_{i-1})$.

We will denote the final (predicted) state by means of π^c with the symbol $s[\pi^c]$ (note that when at least an action turns out to be not applicable, $s[\pi^c]$ is not defined).

In this paper we are interested in the execution of an MMP π in a dynamic environment. In order to accommodate deviations from the expected behavior, at each step of the execution, we have to select on-line a modality for each action such that π^c remains valid for achieving the goal. Formally:

Definition 2 (Multi Modality Plan (Re)configuration) *A (re)configuration problem ψ is the tuple $< \pi^c_i, s'_i, G >$ where: π_i is a suffix of MMP (from the i-th action to the end), s'_i is the world state observed after the execution of $a_{i-1}(m_{i-1})$, G is a set of goal conditions (both propositional and numeric). A solution (if any) is a configuration $c' = \{m'_i, m'_{i+1}, .., m'_{n-1}\} \neq c$ where a modality is selected for each action, and the reconfigured plan $\pi^{c'} = a_i(m'_i), .., a_{n-1}(m'_{n-1})$ is such that: (i) $a_i(m'_i)$ is applicable to s'_i, (ii) each MMA $a_j(m'_j)$ is applicable to s'_j for any $i < j < n$, and (iii) s'_n satisfies G.*

[2] For computational reasons, similarly to what has been done in [11], in this paper we limit ourselves to linear expressions

[3] In our system we support both comparison and propositional goals. So G can be also divided in G_{prop} and G_{num}

Definition 3 (Consistency and Validity) *Given a state s_i and a goal G we say that an MMP π_i and a configuration c is*
–"consistent" if there is at least a solution for the reconfiguration problem $\psi = < \pi_i, s_i, G >$ (i.e. a configuration of action modalities)
–"valid" if c is a solution for ψ

As shown in [12], the reconfiguration can be an effective adaptation mechanism for a continual planning system.

The formulation reported above is relevant w.r.t. the classical formulation of a planning problem, for two main reasons. On the one hand, reasoning about actions and plans involving numeric fluents is very hard and even undecidable when no restrictions are imposed in the used language [10], so a pure replanning approach could represent a barrier for many realistic scenarios. On the other hand, while in many domains the propositional predictions could be quite accurate, it is quite hard to get precise predictions on the effects of actions regarding numeric parameters such as energy, cost and time. As we will see in Section 6, this novel characterization is aimed at providing flexibility and efficiency in handling discrepancies concerning these numeric parameters all along the plan execution, thus avoiding the necessity of (probably expensive) replanning steps.

However, there are open questions: When should the agent activate the reconfiguration mechanism? How do we deal with the reconfiguration from a computational point of view? How useful is the reconfiguration mechanism for adapting the plan execution to the contextual situation? While we will address the third question empirically (see Section 6), we will discuss the other two questions in the following sections.

3 REACTIVE RECONFIGURATION

The problem of deciding when to activate the (re)configuration mechanism can be approached by means of the continual planning paradigm ([1]), straightforwardly extended to deal with a multi-modality plan. Rather than activating the reconfiguration mechanism each time a discrepancy is encountered during the execution (which may be too costly), or intervening just when the next action preconditions are not satisfied (which may be too risky), the reconfiguration is activated only in case π^c becomes invalid.

Algorithm 1 reports the strategy in the specific context of multi modality plans. The plan (with an initial valid configuration) is taken as input by the procedure, which returns *success* (or *failure*) in case the plan has achieved the goals set (or it has not). At each step of the plan execution, the agent observes the environment (line 3), updates its world state representation, and analyzes the plan being executed (line 4). If the current plan configuration is still valid, the iteration proceeds with the execution of the next action from the plan with its instantiated modality. If not, a consistency checking is performed and a possible new configuration is returned (lines 7 to 10).

To handle the problem from a computational point of view, the reconfiguration task is encoded as a Constraint Satisfaction Problem and a Constraint solver can be used for finding a configuration if needed. In particular the CSP encodes variables for modalities and the numeric fluents relevant to the problem.

The modality variables represent the way an action can be executed, so the CSP has a distinct modality variable for each action in π. The numeric fluents variables, on the other hand, aim at capturing the possible trajectories of states behind the MMP, so they have to be replicated as many times as the steps the plan consists of.

In our formulation constraints are implications binding the preconditions and effects of each modality with numeric fluents variables

belonging to the previous (for the preconditions) and successive step (for the effects). Finally, *init* and *goal* constraints restrict the set of reconfigurations to the ones consistent with the current observation acquired (*init* for the initial state), and the goals of the mission.

The CSP solver is invoked to find a new configuration in line 11. If the solver finds a solution, modalities referring to the action already executed remains unchanged, whereas at least one modality of actions still to be executed is changed.

If the plan is not consistent (meaning that there is no solution to the associated reconfiguration problem) a failure is returned[4]. The consistency of the plan is evaluated with the same CSP mechanism.

Algorithm 1: Reactive Reconfiguration

```
   Input: π - Multi Modality Plan, G - goal
   Input: Failure or Success
1  begin
2      while π is not empty do
3          s = observe(environment);
4          if s[π^c] satisfies G then
5              a(m) = π^c.pop();
6              execute (a(m))
7          else
8              if π is not consistent given s and G then
9                  return Failure
10             else
11                 select a c' such that π^c' is valid given s and G;
12                 c = c'
13     s = observe(environment);
14     return s ⊨ G
15 end
```

A limit of this strategy is that the reconfiguration may intervene too late, so that the plan becomes inconsistent and cannot be reconfigured anymore. For instance, if an agent realizes that she is later than expected, although the plan is still valid, it could be necessary to search for a plan accommodating this situation. In the following we will describe a pro-active strategy able to anticipate, at some extent, the potential problem for the plan execution. However, before discussing the new strategy, we will show how to reason on the plan validity in an efficient way.

3.1 Configuration Kernel

As shown in [7, 13], the validity check of a plan involving classical PDDL actions (i.e. without the notion of execution modalities) can be performed by avoiding the prediction/simulation step (i.e. the computation of $s[\pi^c]$): it is sufficient to look at the conditions of the i-th kernel associated to the i-th suffix of the plan still to execute[5]. In fact, a kernel K is a set of propositions and numeric comparisons representing the sufficient and necessary conditions such that, if a state s satisfies each condition in K, the plan leads to the goal from s.

In the case of MMA, the extension is quite simple, as it is possible to take apart the numeric and propositional conditions. Therefore, one can infer if the plan is propositional invalid or just numeric invalid, simply by checking separately the propositional or the numeric conditions involved in K.

[4] In case of failure, it would be possible to invoke a replanner for trying to solve the problem from a generative point of view (see also Section 6)

[5] As shown in [13], the classical notion of kernel [7] can be extended to the numeric case, and also numeric conditions can be constructed via regression starting from the goal statement

We restrict our attention to the case where the original plan π is assumed to guarantee the satisfaction of the propositional goals while we are interested in reasoning about the numeric validity of π^c. In such case we can define:

Definition 4 (Configuration Kernel) *Given a configured plan* $\pi^c = \{a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1}),$ *a state* s *and a goal* G*, a* configuration kernel *is a set of numeric comparisons* $(exp, \{<=, <, ==, >, =>\}, exp')$ *such that* $s[\pi^c] \models G$ *iff* $s \models K$.

The notion of configuration kernel can play a relevant role as made clear by the following proposition.

Proposition 1 *Given a reconfiguration problem* $\psi =< \pi_i, s'_i, G >$*, a configuration* c $\{m_i, m_{i+1}...m_{n-1}\}$ *and a configuration kernel K for* π *instantiated with c, if* s'_i *satisfies K , then c is a solution for* ψ.

Proposition 1 is a direct consequence of definitions 2 and 4; it provides a formal basis for the efficient checking of plan validity. As a matter of facts, it suffices to focus the attention just on the relevant state information, which is the one necessary for verifying if the state satisfies the kernel.

As we will see in the next section, this property is important not only for validity checking purposes, but also for the fact that it provides all the requirements for a correct execution in a very compact form, which can be used also as a basis for more powerful reasoning.

4 PREDICTING ROBUSTNESS VIA CONFIGURATION KERNEL

The formalization provided so far is based on the notion of plan validity. However, more interesting results can be obtained by looking at the problem from a different perspective. In fact, if we give a geometric interpretation to the plan validity, it is easy to see that X(s) and the configuration kernel K respectively represent a point and a validity region inside the vectorial space defined by X. If a state s satisfies (does not satisfy) the kernel, it means that the associated point X(s) is inside (outside) the validity region defined by the kernel.

Given the geometric interpretation, one obvious question concerns how much inside (or how outside) X(s) is. In particular, we can hypothesize that the larger the distance of X(s) from the boundaries defined by the validity region is, the lower the chance to violate (at execution time) the condition in the configuration kernel will be.

For formalizing this intuition, we define a notion of distance which has to capture the contribution of each component in K. Formally:

$$d(X(s), K) = \begin{cases} 0 & \text{if } \exists c' : X(s) \; violates \; c' \\ \min_{c \in K} \frac{d(X(s),c)}{maxD(X,c)} & \text{otherwise} \end{cases} \quad (1)$$

The distance d defined above takes value from 0 to 1. The minimum distance 0 models the situation in which at least a constraint is not satisfied by the current state; this means that the configuration is not valid given that particular state so the configuration has robustness equal to 0. The maximum value 1 is instead reached when, for each constraint involved in the kernel, the current state is at the maximum possible distance. All the intermediate states provide a degree of robustness of a configuration w.r.t. the current state of the world.

The current implementation is restricted to constraints of the form $(a_1x_0 + a_2x_1 +, .., +a_nx_{n-1} + a_0\{<, >\}0)$[6]. Since they are linear combinations, they can be represented as hyperplanes in X, so we can use the euclidean distance as follows:

[6] This kind of representation is possible under the condition of restricting the language to linear expressions, [11]. Numeric constraints involving comparator $\{<=, =>\}$ are treated adding/removing an $\epsilon > 0$.

$$d(X(s), c) = \frac{|a_1 x'_0 + a_2 x'_1 + , .., + a_n x'_{n-1} + a_0|}{\sqrt{a_1^2 + a_2^2 + .. + a_n^2}} \quad (2)$$

where $X(s)$ defining the values $(x'_0, .., x'_{n-1})$ refers to the point in X associated to the state s.

In equation 1, $maxD$ denotes the normalization factor, and it is used to provide a unified scale for all the fluents involved in the problem. We approximate this value exploiting the numeric planning graph ([2, 11]). More precisely, in a preprocessing phase, we perform a reachability analysis starting from the initial state to the goal. This process iteratively produces (the so called) levels containing the propositional atoms and the values of numeric fluents that *could* be reached after the application of a given set of actions. Each level is built by applying only the positive effects of the actions applicable in the previous level. In the case of numeric fluent, the application of the action enlarges the interval of possible values. The iteration is stopped when the process reaches a level satisfying the conditions expressed in the goal, or when the fix point is reached[7].

If the goal is reachable in the planning graph, for each numeric fluent involved in the problem we take the maximum and the minimum value according to the last level, and we use such values, and in particular their geometric interpretation, to figure out the boundaries of X. Having defined such boundaries, the computation of the maximum distance is straightforward.

The distance defined in 1 can be computed for each suffix of the plan. Moreover, under the assumption of not considering assignment operators (in the action model), the distance monotonically decreases towards the goal. Formally:

Proposition 2 *If action modalities in π do not contain assignment operators, for each $i < j$ we have that $d(X(s[\pi_{0\to i}]), K_i) \leq d(X(s[\pi_{0\to j}]), K_j)$*

Formal proof is omitted for lack of space. Let us note that, as an effect of the kernel construction mechanism ([13]), for each $i < j$ the number of constraints present in K_j is always less or equal to the number of constraints involved in K_i; therefore the validity region defined by K_i is strictly smaller than the one defined by K_j.

For this reason, the minimal distance (and so the more critical situation for the whole plan) is the one considering the current state with the kernel associated to the current step of the execution. This is of key importance for the proactive reconfiguration of the next section, as it allows to focus just on a specific kernel.

5 PROACTIVE RECONFIGURATION

Given a consistent plan π, there could be several valid configurations; among them, according to the distance presented in the previous section, one configuration could be "better" than others.

However, searching for the "most robust" configuration may be prohibitive in an execution context, since the computation could require the exploration of the whole space of configurations for a given multi modality plan. In addition, in dynamic environments, the state evolution could often differ from what is expected. So, the optimality of the current configuration all along the task execution it is likely to be compromised, and the effort spent in searching for an optimal solution could be easily nullified.

For these reasons, we adopt a pragmatical approach to the problem. Rather than executing the action in the modality as it has been selected by the current (re)configuration, at each step of the execution the strategy is allowed to change the modality by reasoning on the information provided by the current observation s and the kernel associated to that configuration. As we have seen in the previous section, considering just the current kernel is sufficient to understand the impact of the decision over the whole plan execution.

Algorithm 2: Modality Selection

```
   Input: s - State, π - MMP, G - Goal, a - MMA
   Output: The Selected Modality
1  begin
2      best = mod(a_i);
3      s' = apply(a_i(best));
4      foreach m ∈ mod(a_i) such that m ≠ best do
5          next = apply(a_i(m));
6          if next satisfies K(a_{i+1}(m_{i+1}), .., a_n(m_n),G) then
7              if d(next,K)>d(s',K) then
8                  best = m;
9      return best
10 end
```

Algorithm 2 reports the pseudo-code implementing this idea. In particular the procedure can be invoked from the continual planning algorithm, just after the action is extracted from the plan (Algorithm 1, line 5).

First, the procedure extracts all the alternative modalities of execution from the current action a and performs one step of simulation (line 5) for each of them. Then, the algorithm selects the modality m which maximizes the distance from the predicted state (the one obtained by the one step simulation) to the kernel associated to the arising configuration (the one in which the current action assumes the new modality). Once the modality has been selected, such a decision is reported to the overall continual planning loop so that the agent can execute the action with a modality which is different from what has been planned, adapting the behavior of the plan according to the actual state of the world.

In a few words, the approach computes the consequences of deciding the current modality from a local point of view, but considering what is predicted to happen with the previous configuration. Although the mechanism directly impacts just the way the next action is executed, it could change modalities of several actions. In fact the modality selection procedure is invoked on-line for each action. As we will see in the next section, this approach is crucial for increasing the plan execution success.

6 IMPLEMENTATION AND EVALUATION

In order to evaluate the benefit (and potential drawbacks) of the introduction of the reactive and proactive reconfiguration into a (classical) continual planning architecture (as the one proposed by [1]), we compared the performance of three different architectures:

- LPG-ADAPT (ADP) [8], the basic continual planning architecture of Algorithm 1 where the reconfiguration is substituted with the invocation of LPG-ADAPT.
- RECON-ADAPT (REA-ADP), i.e. the system implementing the reactive configuration strategy (see Algorithm 1), which invokes a reconfiguration supplemented with the invocation of LPG-ADAPT in case the reconfiguration fails.

[7] For details on the numeric extension of the planning graph see [11].

[8] The system has been set with the "-speed" parameter. We noticed that this parameter is crucial for system performance. Note that the plan adaptation is possible by flattening an MMA in more than one PDDL action ([12])

- PROACT-ADAPT (PRO-REA-ADP), the system implementing the proactive strategy (see Algorithm 2). Also in this case, LPG-ADAPT is invoked whenever the reconfiguration fails.

All the systems have hence the replanning capability, but they differ from the presence of the reconfiguration mechanism. We expect that the reconfiguration reduces the need of replanning, hence improving the efficiency of the system. Moreover, we expect that, thanks to the proactive behavior, PRO-REA-ADP could also increase the capacity of completing the plan successfully. This could happen for all those situations where the system would have intervened too late.

The evaluation has been performed into three different numeric domains, extensions of the International Planning Competition domains: Planetary Rover, Zenotravel and DriverLog domain. In particular, we extended the original formulation introducing different execution modalities of the actions where each modality affects the way resources (represented as numeric fluents) are consumed[9]. For instance, the Zenotravel domain already has this feature, since the fly action can be performed either at zoom or cruise mode. In addition we modeled two kinds of boarding (one inexpensive but slower, the other one costly but faster).

	Noise	ADP		REA-ADP		PRO-REA-ADP	
		Cpu-Time	PES	Cpu-Time	PES	Cpu-Time	PES
ZenoTravel	0.1	**101.30**	113	97.56	122	47.79	**147**
	0.15	54.54	59	44.85	62	**67.34**	**111**
	0.2	29.42	32	20.29	29	**56.82**	**79**
	0.25	13.95	14	8.09	10	**50.69**	**59**
	0.3	6.00	6	5.03	6	**24.53**	**28**
	0.35	4.00	4	3.55	4	**18.72**	**21**
	0.4	0.00	0	2.00	2	**11.00**	**11**
	0.45	0.00	0	1.00	1	**6.00**	**6**
	0.5	0.00	0	0.00	0	**4.00**	**4**
	Total	209.21	228	182.38	236	**286.89**	**466**
DriverLog	0.1	65.70	96	75.38	98	**108.63**	**159**
	0.15	45.79	70	53.28	70	**101.40**	**135**
	0.2	45.95	59	36.57	51	**48.11**	**65**
	0.25	37.04	44	**40.82**	**50**	31.73	46
	0.3	**26.73**	29	24.19	**33**	20.89	30
	0.35	18.88	20	**26.76**	**34**	20.92	29
	0.4	**20.27**	**23**	12.53	17	8.64	13
	0.45	**15.57**	16	14.92	**17**	3.67	6
	0.5	10.00	10	**10.76**	**12**	8.23	11
	Total	285.92	367	295.23	382	**352.23**	**494**
PlanetaryRover	0.1	71.88	164	**162.88**	168	37.67	**168**
	0.15	76.81	157	**159.40**	167	62.06	**168**
	0.2	87.60	153	**145.48**	162	77.21	**166**
	0.25	88.51	148	**133.56**	151	80.84	**164**
	0.3	83.41	136	**124.87**	144	78.69	**162**
	0.35	80.11	120	**105.19**	121	80.49	**152**
	0.4	71.52	110	**98.56**	109	85.85	**142**
	0.45	61.04	80	67.26	82	**83.40**	**120**
	0.5	**69.28**	84	54.50	61	55.24	**95**
	Total	690.17	1152	**1051.71**	1165	641.45	**1337**

Table 1. Cpu-time score and number of successful plan executions for each tested systems, in all the considered domains. Results refer to the setup where 5secs are alloted for the computation.

To mimic a real world like scenario, we implemented an environment simulator that returns the actual state of the system obtained by using a noised version of the action model.

To challenge the reconfiguration problem, the noise injection causes unexpected deviations in the consumption of the resources, i.e. the numeric fluents involved in the action. Therefore the plan may become invalid because of the inability of satisfying at least one numeric condition all along the plan (preconditions and/or goals)

We have run[10] each test in nine different settings. In particular, in setting 1, each action consumes 10% more than expected, in setting 2, the noise was increased to 15%, and so on until in setting 9 where the noise was set to 50%.

For all domains, we collected 168 plans (synthesized off-line by using the same LPG-ADAPT system in plan generation modality), which are solutions to problems varying the number of available objects. The length of the resulting plans varies up to 80 actions.

To emulate an on-line context, computational resources devoted to reconfiguration/and or re-planning have to be limited, so the tests have been run in a scenario allotting 5 secs of CPU time for each reconfiguration/replanning task and $2*|\pi|$ seconds as total deliberation time. If a timeout is reached, we consider the plan execution failed.

Performances have been measured considering:

- Plan execution success (PES), i.e. the number of times a given architecture has been able to successfully reach the goal. It is worth noting that in most cases the executed plan differs from the original plan for the changes required by the reconfiguration or the replanning steps; results are showed by Table 1.
- The computation cost, i.e the total amount of CPU time spent to deliberate (which includes the monitoring, modality selection, reconfiguration and replanning). In particular for providing an informative parameter we have used a metric similar to the International Planning Competition metric (http://ipc.icaps-conference.org/). That is, each case submitted is evaluated according to $\frac{T^*}{T}$, where T^* and T are the time spent by the best and the evaluated system, respectively. A not solved case takes 0; results are shown in Table 1.
- Plan completion, i.e. the portion between the number of the actions actually executed by the system and the total number of the actions constituting the plan. This measure provides additional information with respect to PES, since it is able to capture the ability of an architecture to progress in the plan execution; results are shown in Figure 1.

Results show that PRO-REA-ADP is the system exhibiting the maximum competence (PES) over all the domains tested. The advantage is prominent and is estimated around almost 200-250 points, corresponding to the difference of cases solved by PRO-REA-ADP and the other two systems. As concerns the efficiency, REA-ADP behaves quite well in the most of the domains, and is the winner in the Planetary Rover domain. This is explained by the efficiency of the reconfiguration mechanism.

Concerning the average plan completion, PRO-REA-ADP is clearly the system behaving better. However, making exception for the Planetary Rover domain, we do not have a clear winner between LPG-ADP and REA-ADP. Even if we have run a significant set of cases, the stochastic nature of LPG-ADP does not allow an exact comparison between these two systems. We are working on an extension of the experimental setting to better understand this parameter.

In order to evaluate the impact of the timeout on the performance of the system, we have relaxed the maximum cpu-time threshold and we have run the same set of experiments in a scenario allotting 60 secs for the repair and $10*|\pi|$ secs for the total time that can be spent to deliberate. Table 1 reports the PES and CPU Time score for the rover domain. It is easy to see that all the architectures are able to increase PES, that PRO-REA-ADP is still the winner but the differences in performance with respect to LPG-ADAPT are reduced, since the number of times LPG-ADAPT reaches a timeout decreased.

7 DISCUSSION AND CONCLUSION

In recent years the plan execution problem has received an increasing amount of attention and several works [15, 8, 5, 14] have ap-

[9] Visit http://www.di.unito.it/~scala for further details on the domains and the software used.

[10] Experiments ran on a 2.53GHz Intel(R) Core(TM)2 processor with 4 GB, and the Choco solver (www.emn.fr/z-info/choco-solver/) has been used for solving the CSP.

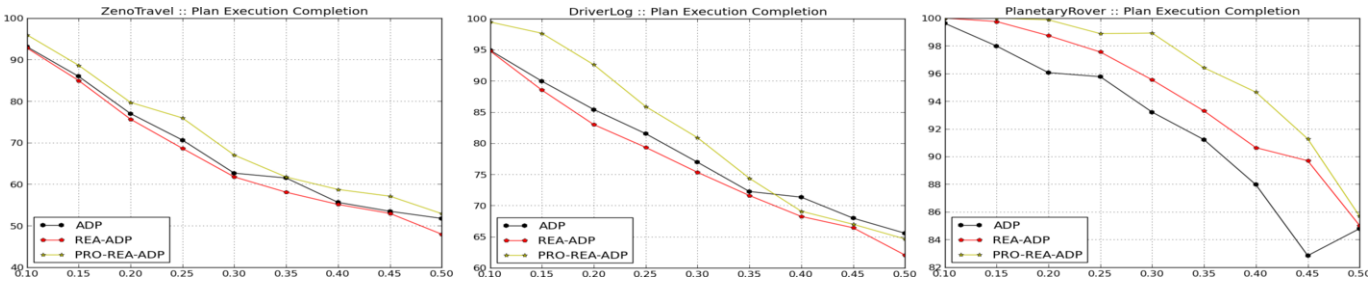

Figure 1. Average plan completion (y-axis) over all the three domains: ZenoTravel, Driverlog and PlanetaryRover. On the x-axis the noise setting.

	Noise	ADP		REA-ADP		PRO-REA-ADP	
		Cpu-Time	PES	Cpu-Time	PES	Cpu-Time	PES
PlanetaryRover (60secs)	0.1	73.04	**168**	161.20	167	40.03	**168**
	0.15	76.73	164	**157.10**	167	60.66	**168**
	0.2	86.52	159	**148.15**	163	76.32	**168**
	0.25	82.47	153	**142.90**	158	79.59	**168**
	0.3	88.15	145	**121.55**	138	82.03	**166**
	0.35	80.30	133	**105.07**	118	85.72	**163**
	0.4	73.41	116	**91.20**	106	86.20	**146**
	0.45	69.82	99	66.63	78	**82.16**	**128**
	0.5	64.09	92	49.65	56	**65.49**	**102**
	Total	694.53	1229	**1043.46**	1151	658.20	**1377**

Table 2. Cpu-time score and number of successful plan executions for all the systems in the Planetary Rover domain. Results refer to the setup where 60secs are alloted for the computation.

proached the problem from a continual planning point of view: the agent executes actions from the plan till it is valid, and activates a re-planning (or adaptation) step when the (unexpected) contextual conditions have threatened the plan. While significant progresses have been made in developing strategies for efficiently handling the plan repair task, few works provide answers for the other questions related to the general problem of continual planning, as for instance deciding when the plan in execution has to be revised. Moreover, not so much attention has been paid to domains where plans are highly constrained in terms of resources usage, in particular when resources are modeled as numeric fluents and the goal contains numerical constraints.

The present paper addresses the problem of plan execution in dynamic environments where discrepancies on the expected resources such as power, fuel, cost and time could compromise the success of the plan. Exploiting a promising new characterization for the repair problem involving resources ([12]), the main focus of this paper is on a proactive strategy aimed at anticipating potential threats to the plan and therefore at reducing possible plan failures. In this context, the closest works to our approach are the ones by [3], [7]. The former proposes an approach for generating branched plans to be used when particular conditions are met in order to opportunistically increase the number of reached goals. As a difference with our approach, the mechanisms presented by Coles operates off-line and addresses the problem of robustness from a probabilistic point of view. The approach presented in [7] works on-line (as in our case) establishing conditions for determining the sub-optimality of the current solution. The main difference with our approach concerns the objective: [7] is mainly concerned in increasing the plan quality, while our approach is aimed at increasing the plan robustness.

The decision of adopting an on line approach has made clear that computational efficiency is a critical aspect. We addressed this problem in a number of ways: first of all we have exploited the notion of multi modality actions where each possible execution modality specifies the impacts/requirements on the numeric fluents representing resources. The strategies have been implemented in a continual planning system combining a CSP encoding and a kernel based formulation ([13]). The former allows to reason about plan reconfigurations once the plan has become invalid (re-action), while the second mechanism provides guidance for a robustness oriented continuous reinforcement of the plan throughout the execution (pro-action).

An experimental analysis on three challenging numeric domains showed the benefit of adopting these two strategies. In particular, the proactive strategy is able to absorb a relevant number of unexpected deviations, reducing (in many situations) the number of plan failures. Moreover, since the plan has to be repaired less frequently, the computational cost for selecting the most adequate execution modality at each step is compensated by a lesser number of reconfigurations or plan adaptations. Finally, both strategies have resulted quite efficient from a computational point of view, hence they can be used in combination with any numeric plan adaptation tool.

The presented approach can be extended in a number of ways. In particular, the proactive mechanism could be extended to deal with a larger set of possible reconfigurations and to perform a trade off between plan robustness and plan quality (possibly by using some of the notions proposed in [7]).

REFERENCES

[1] M. Brenner and B. Nebel, 'Continual planning and acting in dynamic multiagent environments', *Journal of Autonomous Agents and Multiagent Systems*, **19**(3), 297–331, (2009).
[2] A. J. Coles, A. Coles, M. Fox, and D. Long, 'Colin: Planning with continuous linear numeric change', *JAIR*, **44**, 1–96, (2012).
[3] Amanda Jane Coles, 'Opportunistic branched plans to maximise utility in the presence of resource uncertainty', in *Proc. of ECAI-12*, pp. 252–257, (2012).
[4] M. E. desJardins, E. H. Durfee, C. L. Ortiz, and M. J. Wolverton, 'A Survey of Research in Distributed, Continual Planning', *AI Magazine*, **20**(4), (1999).
[5] M. Fox, A. Gerevini, D. Long, and I. Serina, 'Plan stability: Replanning versus plan repair.', in *Proc. of ICAPS-06*, pp. 212–221, (2006).
[6] M. Fox and D. Long, 'Pddl2.1: An extension to pddl for expressing temporal planning domains', *JAIR*, **20**, 61–124, (2003).
[7] C. Fritz and S. A. McIlraith, 'Monitoring plan optimality during execution', in *Proc. of ICAPS-07*, pp. 144–151, (2007).
[8] A. Gerevini and I. Serina, 'Efficient plan adaptation through replanning windows and heuristic goals', *Fundamenta Informaticae*, **102**(3-4), 287–323, (2010).
[9] M. Ghallab, D. Nau, and P. Traverso, 'The actor's view of automated planning and acting: A position paper', *Artificial Intelligence*, **208**(0), 1 – 17, (2014).
[10] Malte Helmert, 'Decidability and undecidability results for planning with numerical state variables', 44–53, (2002).
[11] Jörg Hoffmann, 'The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables', *JAIR*, **20**, 291–341, (2003).
[12] E. Scala, R. Micalizio, and P. Torasso, 'Robust execution of rover plans via action modalities reconfiguration', in *Proc. of ICAART-14*, pp. 142–152, (2014).
[13] Enrico Scala, 'Numeric kernel for reasoning about plans involving numeric fluents', volume 8249 of *LNCS*, 263–275, (2013).
[14] Enrico Scala, 'Plan repair for resource constrained tasks via numeric macro actions', in *(to appear) Proc. of ICAPS-14*, (2014).
[15] van der Krogt R. and de Weerdt M., 'Plan repair as an extension of planning', in *Proc. of ICAPS-05*, pp. 161–170, (2005).